Option Explicit

'Last Revision January 12, 2015

Public Const PUB\_YAHOO\_URL\_STR As String = "http://real-chart.finance.yahoo.com/table.csv?s=AAPL&d=0&e=12&f=2015&g=d&a=11&b=12&c=1980&ignore=.csv" 'for Lesson 2.1.1: Constants

Type QuoteData ' for Lesson 2.1.5 Types in VBA

dtDate As Date

m\_Open As Double

m\_High As Double

m\_Low As Double

m\_Close As Double

lngVolume As Long

End Type

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2: Value Types And Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Most financial programming involves making mathematical calculations. As in algebra, we often use variables in computer programs to hold

' different values we need for calculation. In this section, you will learn how to declare variables and perform calculations in VBA.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.1: Declaring Variables

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' To a computer, primitive or simple value types, called variables, are actual, physical spaces in memory that store data for use by our

' program. Before we can use a variable, we need to declare it using the Dim statement. That is, we have to tell the computer to set up a

' space in memory with a specific name. In programming, the variable names we use are usually descriptive of the contents they hold. For

' example, a program to analyze stock returns might contain variables like this:

' Dim m\_MondayClose

' Dim m\_StockPrice As Integer

' Dim m\_CallDelta As Double

' These lines of code set up variables, physical places in memory, that will be known by the names m\_MondayClose, m\_StockPrice, m\_CallDelta

' and strTicker. Furthermore, the types of data that will go into each of these containers will be things called a single, a double and a

' string. Single, double and string are value types, which tell us what kind of data the variable can hold. Here is a list of the different

' value types supported by VBA, with descriptions:

' Value Type with Identifier: Boolean

' Range: True or False

' Note: 16 bits. Stored internally as an either 0 or 1.

' Example Using Naming Convention and Value Type Identifier: Dim m\_BuySell As Boolean

' Value Type with Identifier: Date

' Range: 1/1/01000 to 12/31/9999

' Note: 64 bits. Holds dates and times.

' Example Using Naming Convention and Value Type Identifier: Dim m\_ExpDate As Date

' Value Type with Identifier: Decimal

' Range: 1.0E-28 to 7.9E+28

' Note: 128 bits. Large numbers.

' Example Using Naming Convention and Value Type Identifier: Dim m\_Covar As Decimal

' Value Type with Identifier: Double

' Range: +/-5.0E-324 to +/-1.7E+308

' Note: 64 bits. Double- precision floating- point variable.

' Example Using Naming Convention and Value Type Identifier: Dim m\_CallDelta As Double

' Value Type with Identifier: Integer

' Range: -2,147,483,648 to 2,147,483,647

' Note: 32 bits. Integers only. No decimal numbers.

' Example Using Naming Convention and Value Type Identifier: Dim m\_NumShares As Integer

' Value Type with Identifier: Long

' Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

' Note: 64 bits. Big integers, but still no decimal numbers.

' Example Using Naming Convention and Value Type Identifier: Dim m\_NumTrades As Long

' Value Type with Identifier: String

' Range: Varies based upon the number of characters.

' Note: Character Data.

' Example Using Naming Convention and Value Type Identifier: Dim strTicker as String

' Value Type with Identifier: Object

' Range: Varies based upon the Object reference.

' Note: Object variables are stored as 32-bit (4-byte) addresses that refer to objects.

' Example Using Naming Convention and Value Type Identifier: Dim mPortfolio as Object

' Value Type with Identifier: Variant

' Range: Varies

' Note: A special data type that can contain numeric, string, or date data as well as the special values Empty and Null

' The Variant data type has a numeric storage size of 16 bytes and can contain data up to the range of a Decimal,

' or a character storage size of 22 bytes (plus string length), and can store any character text.

' The VarType function defines how the data in a Variant is treated. All variables become Variant data types if not

' explicitly declared as some other data type.

' Example Using Naming Convention and Value Type Identifier: Dim mData as Variant

'When a variable of any type is created, its default value is 0. We can define, or change the values

'of our variables this way: m\_MondayClose = 10. In VBA, we CANNOT declare and define a variable

'in the same line: Dim m\_StockPrice as Double = 4.92. Thanks to Ryan Hojeong Seo for his feedback!

'In VBA all variables should be declared before they can be used. Later in the course, I will show

'you that this helps avoid common programming errors.

'-----------------------------------------------------------------------------------------------------------------------------------

'EX1: Following are the default initial values for the more popular data types in VBA:

'-----------------------------------------------------------------------------------------------------------------------------------

' Byte \_ 0

' Boolean \_ False

' Integer \_ 0

' Long \_ 0

' Single \_ 0

' Double \_ 0

' Currency \_ 0

' Decimal \_ 0

' Date \_ 12/30/1899 12:00:00 AM

' Object \_ Nothing

' String \_ ""

' Variant \_ Missing

'You can verify this by calling the following function and checking the Immediate Window.

Sub TEST\_DEFAULT\_VARIABLE\_PARAMETERS()

' Please notice that there are times when we may not be required to pass all the arguments in a parameter list to a procedure. This

' is typically the case when parameters later in the list are dependent on specific values of variables earlier in the list. To declare

' a parameter as optional, we include the Optional keyword in the parameter declaration. When we declare a parameter as optional, all

' subsequent parameters in the list must also be optional.

Call DEFAULT\_VARIABLE\_PARAMETERS

End Sub

Function DEFAULT\_VARIABLE\_PARAMETERS(Optional byteVar As Byte, \_

Optional boolVar As Boolean, \_

Optional IntVar As Integer, \_

Optional longVar As Long, \_

Optional singleVar As Single, \_

Optional doubleVar As Double, \_

Optional currencyVar As Currency, \_

Optional dateVar As Date, \_

Optional objVar As Object, \_

Optional stringVar As String, \_

Optional VariantVar As Variant)

Dim decimalVar As Variant

' print default values for each data type

Debug.Print "Byte default value is " & byteVar

Debug.Print "Boolean default value is " & boolVar

Debug.Print "Integer default value is " & IntVar

Debug.Print "Long default value is " & longVar

Debug.Print "Single default value is " & singleVar

Debug.Print "Double default value is " & doubleVar

Debug.Print "Currency default value is " & currencyVar

' decimal

decimalVar = CDec(decimalVar)

Debug.Print "Decimal default value is " & decimalVar

Debug.Print "Date default value is " & \_

Format$(dateVar, "Short Date") & " " & Format$(dateVar, "Long Time")

Debug.Print "Object default value is Nothing? " & (objVar Is Nothing)

Debug.Print "String default value is " & stringVar

Debug.Print "Variant default is Missing? " & IsMissing(VariantVar)

End Function

'What this means is that in certain situations, it is no longer required to declare default values for certain variables.

'Boolean variables are a popular target for this technique.

'Suppose you have a procedure that accepts a boolean value which by default should be False. You might declare the function

'like this:

'Function DoSomething(Optional myVar As Boolean = False)

'Since the default initial value of a boolean variable is False, we don't need to declare the function like this. We can

'simply write: Function DoSomething(Optional myVar As Boolean). If not passed by argument, myVar will be False by default.

'-----------------------------------------------------------------------------------------------------------------------------------

'EX 2: Test Multi Type

'-----------------------------------------------------------------------------------------------------------------------------------

Private Sub TestMultiType()

Dim b1 As Byte

Dim b2 As Byte

Dim i1 As Integer

Dim i2 As Integer

Dim l1 As Integer

Dim l2 As Integer

Dim s1 As Single

Dim s2 As Single

Dim d1 As Double

Dim d2 As Double

Dim str1 As String

Dim str2 As String

'Reference: http://msdn.microsoft.com/en-us/library/47zceaw7(VS.80).aspx

On Error GoTo ERROR\_LABEL

b1 = 5: b2 = 7

i1 = 5: i2 = 7

l1 = 5: l2 = 7

s1 = 5: s2 = 7

d1 = 5: d2 = 7

str1 = "wer": str2 = "ert"

'vbCrLf = Chr(13) + Chr(10) = Carriage return\_linefeed combination

MsgBox \_

"Byte: " & MAX\_FUNC(b1, b2) & vbCrLf & \_

"Integer: " & MAX\_FUNC(i1, i2) & vbCrLf & \_

"Long: " & MAX\_FUNC(l1, l2) & vbCrLf & \_

"Single: " & MAX\_FUNC(s1, s2) & vbCrLf & \_

"Double: " & MAX\_FUNC(d1, d2) & vbCrLf & \_

"String: " & MAX\_FUNC(str1, str2)

Exit Sub

ERROR\_LABEL:

End Sub

Private Function MAX\_FUNC(ByVal FIRST\_VAL As Variant, \_

ByVal SECOND\_VAL As Variant) 'returns variant

'Try replacing the variant types below with double, also add as Double on the end of

'the next line to return a double this should speed it up

On Error GoTo ERROR\_LABEL

If SECOND\_VAL > FIRST\_VAL Then

MAX\_FUNC = SECOND\_VAL

Else

MAX\_FUNC = FIRST\_VAL

End If

Exit Function

ERROR\_LABEL:

MAX\_FUNC = Err.Number

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2.1.1: Variable Scope

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Variables and constants can also be declared using an access modifier. Access modifiers serve to specify the scope and accessibility of

'the variable. The access modifiers are Private and Public. Here is an example: Dim m\_Price as Double

'In the next lesson, we will discuss access and scope in more detail. For now, be aware that the scope of a variable refers to the

'parts of a program that can access a variable. Not all variables are accessible everywhere. Variables in Visual Basic can have the

'following scope:

'Scope: Module

'Accessibility or "Visibility": Accessible to all functions and procedures defined in the module.

'Scope: Class

'Accessibility or "Visibility": Accessible in what is known as the declaration space of the class.

'Scope: Global

'Accessibility or "Visibility": Accessible anywhere in a project.

'Variables should always be defined with the smallest possible scope. Variables with global scope can make the logic of an application

'extremely difficult to understand and make the reuse and maintenance of your code more difficult. In a Visual Basic application,

'global variables should be used only when there is no other convenient way to share data between parts of your program. When global

'variables must be used, it is good practice to declare them all in a single module, grouped by function.

'For variables it is also possible to declare them at the top of a module so that they will be available anywhere within

'the module (if declared as Private) or anywhere within the project (if declared as Public). But remember that any change

'to a variable will be affect every procedure where the variable is used, and this may not be what was intended.

'A variable of the same name may be declared within any procedure without any warning being generated. Changes to this

'variable will only will only be seen within the procedure where it is declared, but it may be assumed (especially at a later date)

'that the variable with global scope is applicable in every routine.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2.1.2: Constants

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'If the value of a variable is not going to change over the life of our program, we should declare it as a constant rather than a

'variable like this:

'Public Const PUB\_YAHOO\_URL\_STR As String = "http://real-chart.finance.yahoo.com/table.csv?s=AAPL&d=0&e=12&f=2015&g=d&a=11&b=12&c=1980&ignore=.csv" 'for Lesson 2.1.1: Constants

'Declaring a value as a constant protects it against accidentally being changed down the road.

'Please notice that by default, when a variable or constant is declared within a VBA subroutine or function it is only available

'within that routine. To get the YAHOO information available you need to use the variable PUB\_YAHOO\_URL\_STR

'at the time of calling the sub. For example,

Sub RUN\_CONSTANTS\_YAHOO\_EXAMPLE()

Application.ScreenUpdating = False 'True if screen updating is turned on. Speed Up Excel!

Workbooks.Open PUB\_YAHOO\_URL\_STR 'Make sure you have an internet connection

Application.ScreenUpdating = True

End Sub

'Since the value of PUB\_YAHOO\_URL\_STR will not be changed during the execution of the RUN\_CONSTANTS\_YAHOO\_EXAMPLE routine, it generally makes

'sense to give it the widest scope possible. That is declaring it as Public at the top of this module. In this way the constant will be available to every module

'in the project. For large projects it is a good idea to put all constants in their own module.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2.1.3: Representing Dates and Times

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' When making calculations, we also often need to represent dates and times in our programs for things like interest accrual and

' trade time stamps.

Sub TEST\_DATES\_TIMES()

Dim m\_Date As Date

m\_Date = Now()

m\_Date = DateSerial(Year(m\_Date), Month(m\_Date), Day(m\_Date))

Debug.Print m\_Date

End Sub

' Dim m\_Date As DateTime

' m\_Date = Now()

'VBA is sensitive to the cultural differences in date representation. For example, if you are working in the United Kingdom and

'rerun the above example, the first four dates are interpreted as the 2nd of January, rather than the 1st of February.

'On the Object Browser (on the keyboard F2 or on the View Menu look for Object Browser) look for the function DateDiff, and take

'a look at the inputs.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2.1.4: Option Explicit

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'In VBA an Option Explicit statement should always appear in the declarations section of a module. Option Explicit prevents

'Visual Basic from making implicit type conversions that may involve loss of data. For your assignments you should always have the

'Option Explicit statement at the top in your programs.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 2.1.5: Types in VBA

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Generally, when a group of data fit together, but consist of different value types, we may prefer to create our own variable type,

'using a Type. Visual Basic allows us to create our own user-defined value types using the Type statement. Our Type

'will generally contain more than one element and each element must be declared with an access modifier at the top of the module.

'Here is an example of a user-defined data type called QuoteData:

'Type QuoteData

' dtDate As Date

' m\_Open As Double

' m\_High As Double

' m\_Low As Double

' m\_Close As Double

' lngVolume As Long

'End Type

'We can then declare a variable of the type QuoteData in the following way:

Private Sub Test\_Quote\_Date()

Dim m\_StockPrice As QuoteData

m\_StockPrice.dtDate = Now()

m\_StockPrice.m\_Open = 324.54

Debug.Print Format(m\_StockPrice.dtDate, "dd-mm-yy"), Format(m\_StockPrice.m\_Open, "$0.0")

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2: Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'VBA has a wealth of operators to handle mathematical calculations and other logical operations. As we go through the class, we will

'be making extensive use of operators as we write programs. Most of them are self-explanatory, but some may not be. You can use this

'section as a reference as they come up over the course of the class.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2.1: Arithmetic Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub Arithmetic\_Operators()

Dim x As Double: Dim y As Double

x = 4: y = 2

'Exponentiation: ^

Debug.Print x ^ y 'Raises x to the power of y

'Negation: -

Debug.Print -y 'Negates y

'Multiplication: \*

Debug.Print x \* y 'Multiplies x And y

'division: /

Debug.Print x / y 'Divides x by y and returns a floating point result

'Integer division: \

Debug.Print x \ y 'Divides x by y and returns an integer result

'Modulos: Mod

Debug.Print x Mod y 'Divides x by y and returns the remainder

'Addition: +

Debug.Print x + y 'Adds x And y

'Subtraction: -

Debug.Print x - y 'Subtracts y from x

Debug.Print 3 Mod 2

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2.2: Comparison Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub Comparison\_Operators()

Dim x As Double: Dim y As Double

y = 5.67

x = 4 'Equal

Debug.Print x <> y 'Not Equal

Debug.Print x > y 'Greater Than

Debug.Print x < y 'Less Than

Debug.Print x >= y 'Greater Than Or Equal

Debug.Print x <= y 'Less Than Or Equal

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2.3: Assignment Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub Assignment\_Operators()

Dim x As Double

x = 10

x = x + 3 '+=

Debug.Print x

x = x - 2 '-=

Debug.Print x

x = x \* 1.15 '\*=

Debug.Print x

x = x / 2 '/=

Debug.Print x

x = x \ 3 '\=

Debug.Print x

x = x ^ 0.2 '^=

Debug.Print x

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2.4: Logical Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub Logical\_Operators()

Debug.Print (4 > 5) And (5 < 9) 'Evaluates to True only if both conditions are true.

Debug.Print (4 > 5) Or (5 < 9) 'Evaluates to True if one or both conditions are true.

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.2.5: Concatenation Operators

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub NICO()

Dim i As Long

Dim j As Long

For i = 1 To 10

For j = 1 To 5

Debug.Print "i = " & i & " ,j= " & j

Next i

Next j

End Sub

Sub Concatenation\_Operators()

Dim s As String

s = "NICO"

s = s & "LAS" '& = concatenates or binds a number of strings together.

Debug.Print s

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.3: Control Structures

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'The code we have written thus far has all been linear, or sequential, in nature. That is, lines of code were executed in order,

'one after the other, till the end of the program. Although this is fine for very short tasks, to tackle more complex situations,

'we will need to employ control structures, which involve the use of program flow statements. Program flow statements fall into

'one of two categories:

'1) Decision Structures. Conditional statements in which code is executed based on whether or not a condition is met.

'2) Repetition Structures. Looping statements, in which code is executed repeatedly either a number of times or until a condition

'is met.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.3.1: If...Then...Else Statement

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'The If...Then...Else statement lets us say, in effect, "If this is true, then do this; otherwise, do that." The logic couldnêt be

'more intuitive. The following example illustrates the use of the If...Then...Else structure.

Function Control\_Structure\_If\_Then\_Else\_Statement()

Dim Signal\_Str As String

Dim m\_StockPrice As Double

m\_StockPrice = 60

If m\_StockPrice > 55 Or m\_StockPrice < 40 Then

Signal\_Str = "SELL"

Else

Signal\_Str = "HOLD"

End If

Control\_Structure\_If\_Then\_Else\_Statement = Signal\_Str

'Debug.Print Signal\_Str

End Function

'In the example, the statements following the If are executed only if the expression evaluates to True, that is if the stock price is

'greater than 55 or less than 40. The Else block of code executes if the expression evaluates to False. So, in this case, if the stock

'price is between 40 and 55, we will hold. The expression used in If...Then is a Boolean expression, true or false. The use of the Else

'block in an If statement is optional.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.3.2: The Select...Case Statement

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'The Select...Case structure is very similar to the If...Then...Else structure, but it is much more efficient and makes our code much

'more readable if there are several branches to the decision structure. In the Select...Case structure we can include an unlimited number

'of clauses. Letês look at an example that not only illustrates the logic statements within a Select...Case framework, but also demonstrates

'how to build a histogram of log returns:

Sub Control\_Structure\_The\_Select\_Case\_Statement()

Dim m\_Return As Double

Dim m\_Bin1, m\_Bin2, m\_Bin3, m\_Bin4, m\_Bin5, m\_Bin6 As Integer

m\_Return = Math.Log(51 / 50)

Select Case m\_Return

Case Is < -0.02

m\_Bin1 = m\_Bin1 + 1

Case -0.02 To -0.01

m\_Bin2 = m\_Bin2 + 1

Case -0.01 To 0

m\_Bin3 = m\_Bin3 + 1

Case 0 To 0.01

m\_Bin4 = m\_Bin4 + 1

Case 0.01 To 0.02

m\_Bin5 = m\_Bin5 + 1

Case Is > 0.02

m\_Bin6 = m\_Bin6 + 1

Case Else

GoTo 1983

End Select

Debug.Print m\_Bin5

'Since the natural log of ( 51 / 50 ) is .0198, the value of m\_Bin5 will be incremented by 1. The Case Else clause at the

'end of the structure is optional. Also, multiple conditions are evaluated separately with a logical OR as opposed to an

'AND, so itês best to keep Select...Case logic as simple as possible. Letês look at another example evaluating strings.

'Call and put option symbols include a strike price and expiration month. The second to last letter in the symbol denotes

'the month of expiration and the last term the price. So, for example, GEKD would be the symbol for the General Electric

'November 20.00 calls. GERT would be the June 17.50 puts. We will have more examples using option symbols later in the

'class, but here is a Select...Case structure using the char data type to determine the month of expiration:

Dim m\_Month As String

m\_Month = "D"

Select Case m\_Month

Case "A", "a", "M", "m"

m\_Month = "January"

Case "B", "b", "N", "n"

m\_Month = "February"

Case "C", "c", "O", "o"

m\_Month = "March"

Case "D", "d", "P", "p"

m\_Month = "April"

Case "E", "e", "Q", "q"

m\_Month = "May"

Case "F", "f", "R", "r"

m\_Month = "June"

Case "G", "g", "S", "s"

m\_Month = "July"

Case "H", "h", "T", "t"

m\_Month = "August"

Case "I", "i", "U", "u"

m\_Month = "September"

Case "J", "j", "V", "v"

m\_Month = "October"

Case "K", "k", "W", "w"

m\_Month = "November"

Case "L", "l", "X", "x"

m\_Month = "December"

Case Else

GoTo 1983

End Select

Debug.Print m\_Month

'Since the value of chrMonth is "D," the value of strMonth will be set to "April."

Exit Sub

1983:

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.4: Repetition Structures

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'VBA provides a number of different types of loops that you can use to implement repetitive operations.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' Lesson 2.4.1: The For...Next Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'The For...Next loop executes a series of statements a specific number of times. The basic syntax is:

Sub Repetition\_Structures\_For\_Next\_Loop()

Dim i As Integer

For i = 0 To 10 Step 2

Debug.Print "Your stock is down " & i & " points."

Next i

'Here, the program will loop through this code five times, starting with x = 0. Each time it loops x will

'incremented by two until the maximum value of x, in this case 10, is reached. In the example above, the

'printout will show our stock fall by 2 points with each successive loop. If the Step phrase is left out,

'your program will automatically increment the loop counter variable by +1. Letês take a look at this code:

Dim s As Integer

For i = 1 To 5

s = s + i 'Lesson 2.2.3: Assignment Operators

Next i

Debug.Print s

'After completing the loop, the value of s = 1 + 2 + 3 + 4 + 5 = 15

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.4.2: The Do While Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Here is an example of a Do...While Loop.

Sub Repetition\_Structures\_Do\_While\_Loop()

Dim m\_StockPrice As Double

m\_StockPrice = 35

Do While m\_StockPrice < 100

m\_StockPrice = m\_StockPrice + 1

Debug.Print m\_StockPrice

Loop

End Sub

'When this loop is finished, it prints out the price as 100. This routine evaluates m\_StockPrice < 100 each time through the loop.

'When m\_StockPrice = 99, the loop increments m\_StockPrice to 100. The next evaluation of m\_StockPrice = 100 is False so program

'execution exits the loop and continues with the line after the Loop statement, printing m\_StockPrice as 100.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.4.3: The Do...Until Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Here is an example of a Do...Until Loop.

Sub Repetition\_Structures\_Do\_Until\_Loop()

Dim m\_SellPrice As Double

m\_SellPrice = 95

Dim m\_StockPrice As Double

m\_StockPrice = 45

Do Until m\_StockPrice >= m\_SellPrice

m\_StockPrice = m\_StockPrice \* Math.Exp(0.1)

Debug.Print "We are still holding the stock."

Loop

Debug.Print "We have sold the stock at " & Format(m\_StockPrice, "$0.00")

End Sub

'As with the Do...While loop, the Do...Until is not necessarily executed at all since the program evaluates the exit

'condition before entering the loop. In this example, we sold the stock at 100.149.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.4.4: The Do...Loop While Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'To make sure that a loop executes at least once, place the exit condition at the Loop statement, rather than at the Do statement,

'as in the following:

Sub Repetition\_Structures\_Do\_Loop\_While\_Loop()

Dim m\_StockPrice As Double

m\_StockPrice = 35

Do

Debug.Print "Decrementing the stock price."

m\_StockPrice = m\_StockPrice - 1

Loop While m\_StockPrice > 30

Debug.Print "Sold the stock at " & Format(m\_StockPrice, "$0.00")

End Sub

'In this program the stock is sold at 30.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.4.5: The Do... Loop Until Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'You can similarly put the Until condition at the end of a loop. In the previous example you knew you wanted to go through the loop

'at least once. By putting the Until statement at the end, you don't need to worry about the initial value of the variable.

Sub Repetition\_Structures\_Do\_Loop\_Until\_Loop()

Dim m\_StockPrice As Double

m\_StockPrice = 35

Do

m\_StockPrice = m\_StockPrice - 1

Loop Until m\_StockPrice = 25

Debug.Print "We sold the stock at " & Format(m\_StockPrice, "$0.00")

End Sub

'In this program, the stock is sold at 25.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.4.6: The While...End While Loop

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'VBA also provides another general-purpose loop statement called the While ...End While loop. The While...End While loop

'has the following syntax:

Sub Repetition\_Structures\_While\_End\_While\_Loop()

Dim m\_StockPrice As Double

m\_StockPrice = 35

Do While m\_StockPrice <= 50

m\_StockPrice = m\_StockPrice + 1

Debug.Print "Holding the stock."

Loop

Debug.Print "We sold the stock at " & Format(m\_StockPrice, "$0.00")

End Sub

'In this program the stock is sold at 51.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.5: The Exit Commands

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'There are occasions where you need to break out of a loop. In such a case we can insert an Exit command. Depending on which

'type of loop structure you are using, you will use the Exit For command, or the Exit Do command. We might generally do this

'inside an If...Then statement inside a loop. Here is an example of an infinite loop. The Do While 1 statement will never

'evaluate to False, so this program will loop forever until some event causes an exit from the loop. As you can imagine,

'its best to be very careful with infinite loops.

Sub Exit\_Commands()

Dim m\_StockPrice As Double

m\_StockPrice = 35

Do While 1

m\_StockPrice = m\_StockPrice + 1

If m\_StockPrice > 100 Then

Exit Do

End If

Loop

Debug.Print "We sold the stock at " & Format(m\_StockPrice, "$0.00")

End Sub

'In this program, we sold the stock at 101.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.6: Nested Loops

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'You can put a For...Next loop inside of another For...Next loop. Consider the following example showing nested For...Next loops to

'transpose a matrix. Again, we havenêt looked at arrays yet, so donêt worry about the variable references. For now, just note the

'structure of embedded loops.

Sub Nested\_Loops()

Dim i, j As Integer

Dim NROWS As Integer

Dim NCOLUMNS As Integer

NROWS = 10

NCOLUMNS = 5

For j = 1 To NCOLUMNS

For i = 1 To NROWS

Debug.Print i, j

Next i

Next j

End Sub

'Although For...Next loops are useful when we know in advance how many times we want to execute the loop, there are occasions when we do

'not have this information in advance.

'Example: Testing the speed of different data types with nested loops!!

Private Sub NESTED\_LOOPS\_TESTING\_SPEED\_DATA\_TYPES()

Dim i As Single

Dim j As Single

Dim T\_VAL As Single

Dim tByte0 As Byte

Dim tByte1 As Byte

Dim tByte2 As Byte

Dim tByte3 As Byte

Dim tInt0 As Integer

Dim tInt1 As Integer

Dim tInt2 As Integer

Dim tInt3 As Integer

Dim tLong0 As Long

Dim tLong1 As Long

Dim tLong2 As Long

Dim tLong3 As Long

Dim tSingle0 As Single

Dim tSingle1 As Single

Dim tSingle2 As Single

Dim tSingle3 As Single

Dim tDouble0 As Double

Dim tDouble1 As Double

Dim tDouble2 As Double

Dim tDouble3 As Double

Dim tVariant0 As Variant

Dim tVariant1 As Variant

Dim tVariant2 As Variant

Dim tVariant3 As Variant

i = 254

j = 200

'Mod Syntax

'Returns the remainder after number is divided by divisor.

'The result has the same sign as divisor.

'Example: number MOD divisor

'Number: is the number for which you want to find the remainder.

'Divisor: is the number by which you want to divide number.

'test byte

T\_VAL = Timer

For tByte0 = 1 To i

For tByte1 = 1 To i

For tByte2 = 1 To i

tByte3 = (tByte2 + tByte1 / tByte0) Mod j

Next tByte2

Next tByte1

Next tByte0

T\_VAL = Timer - T\_VAL

Debug.Print "test byte: " & T\_VAL

'test integer

T\_VAL = Timer

For tInt0 = 1 To i

For tInt1 = 1 To i

For tInt2 = 1 To i

tInt3 = (tInt2 + tInt1 / tInt0) Mod j

Next tInt2

Next tInt1

Next tInt0

T\_VAL = Timer - T\_VAL

Debug.Print "test integer: " & T\_VAL

'test long

T\_VAL = Timer

For tLong0 = 1 To i

For tLong1 = 1 To i

For tLong2 = 1 To i

tLong3 = (tLong2 + tLong1 / tLong0) Mod j

Next tLong2

Next tLong1

Next tLong0

T\_VAL = Timer - T\_VAL

Debug.Print "test long: " & T\_VAL

'test single

T\_VAL = Timer

For tSingle0 = 1 To i

For tSingle1 = 1 To i

For tSingle2 = 1 To i

tSingle3 = (tSingle2 + tSingle1 / tSingle0) Mod j

Next tSingle2

Next tSingle1

Next tSingle0

T\_VAL = Timer - T\_VAL

Debug.Print "test single: " & T\_VAL

'test double

T\_VAL = Timer

For tDouble0 = 1 To i

For tDouble1 = 1 To i

For tDouble2 = 1 To i

tDouble3 = (tDouble2 + tDouble1 / tDouble0) Mod j

Next tDouble2

Next tDouble1

Next tDouble0

T\_VAL = Timer - T\_VAL

Debug.Print "test double: " & T\_VAL

'test variant

T\_VAL = Timer

For tVariant0 = 1 To i

For tVariant1 = 1 To i

For tVariant2 = 1 To i

tVariant3 = (tVariant2 + tVariant1 / tVariant0) Mod j

Next tVariant2

Next tVariant1

Next tVariant0

T\_VAL = Timer - T\_VAL

Debug.Print "test variant: " & T\_VAL

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'2.7: Lesson Summary

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'In this lesson you have been exposed to all the different variable types available in VBA. Also, you should now understand how to

'declare variables using the Dim statement and the various identifiers and access modifiers as well as how to define them. You should

'also understand the importance of the Option Explicit. Also, our variable naming convention require that we add a prefixes to our

'variable names that indicate the data type of the variable. Variable names should also describe the something about the nature of the

'value such as m\_StockPrice.

'Further, we looked at the different operators available in VBA and how some of them could be used. We also learned how to use

'If..Then..Else statements, Select Case statements, and many different kinds of loops to control program flow. Loops will become

'more important in future lectures about arrays and data structures.